SplayNet: Distributed User-Space Topology Emulation

Valerio Schiavoni, Etienne Rivière, and Pascal Felber

University of Neuchâtel, Switzerland

Abstract. Network emulation allows researchers to test distributed applications on diverse topologies with fine control over key properties such as delays, bandwidth, congestion, or packet loss. Current approaches to network emulation require using dedicated machines and low-level operating system support. They are generally limited to one user deploying a single topology on a given set of nodes, and they require complex management. These constraints restrict the scope and impair the uptake of network emulation by designers of distributed applications. We propose a set of novel techniques for network emulation that operate only in user-space without specific operating system support. Multiple users can simultaneously deploy several topologies on shared physical nodes with minimal setup complexity. A modular network model allows emulating complex topologies, including congestion at inner routers and links, without any centralized orchestration nor dedicated machine. We implement our user-space network emulation mechanisms in SplayNet, as an extension of an open-source distributed testbed. Our evaluation with a representative set of applications and topologies shows that SplayNet provides accuracy comparable to that of low-level systems based on dedicated machines, while offering better scalability and ease of use.

Keywords: Topology emulation, large-scale networks, testbeds.

1 Introduction

A key aspect of distributed systems evaluation is the capacity to deterministically reproduce experiments and compare distributed applications in the same deployment context, and in particular when operating under the same network conditions. Distributed testbeds such as PlanetLab (`www.planet-lab.org`) allow testing applications in real-world conditions, by aggregating a large number of geographically distant machines. While extremely useful for large-scale systems evaluation, such testbeds cannot be reconfigured to expose a variety of network infrastructures or topologies. Furthermore, the high load and the unpredictable running conditions of shared testbeds are a hindrance for the reproducibility of evaluation results, or for the fair comparison of different applications.

Network emulation supports *controllable* and *reproducible* distributed systems evaluation. It allows running a distributed application on dedicated machines as if it were running on an arbitrary network topology, and observe the behavior of the application in various network conditions. The emulation of communication

D. Eyers and K. Schwan (Eds.): Middleware 2013, LNCS 8275, pp. 62–81, 2013.

links is based on an input *topology*, i.e., a graph representation of nodes, routers, and the properties of their connections. A cluster with a high-performance local network can typically support the execution of applications and the emulation of topologies.

The focus of this paper is on providing support for easy evaluation of networked applications (e.g., indexing [35], streaming [5], coding [15], data processing over non-standard topologies [11], etc.) under diverse yet reproducible networking conditions. Furthermore, we seek to provide support for concurrent deployments of emulated topologies and distributed applications, where the physical nodes of a cluster can be used for running multiple experiments with different topologies, without interference and loss of accuracy for any of the experiments. Finally, we posit that the uptake of network emulation mechanisms will be greater if the setup of such mechanisms remain simple and cross-platform, and if they are integrated with a toolkit that facilitates distributed systems prototyping and evaluation, for researchers, students, and engineers. This requires mechanisms and tools for rapid development, deployment, observation, and control of distributed experiments. Note that our work focuses on the evaluation of networked applications on top of standard TCP and UDP connections, when presented with various end-to-end characteristics: bandwidth, delay, packet loss, and congestion. We do not consider the evaluation of the network stack itself, or the evaluation of low-level network characteristics and protocols, which is the focus of other tools [23].

Existing solutions [1,7,16,18,19,26,28–30,33,34,38–40] support emulation of part or all of the characteristics of a topology, but present a number of limitations. None allows researchers to deploy several network topologies *at the same time* and *on the same physical nodes* over a shared platform. Indeed, they enforce that a node of the testbed is used by one user, for one topology: this requires a large amount of physical resources, or imposes severe restrictions on the number of users and/or the size of their experiments. Furthermore, existing approaches require privileged or *root* access to the machines of the testbed, and often the use of dedicated machines or specialized operating systems to support network emulation. Finally, most of them require to completely reconfigure testbed nodes for every new emulated topology.

Contributions. The main contributions of this paper are the following:

- We propose a novel approach for supporting network emulation with user-space mechanisms and without support from the operating system. Our approach allows emulating complex topologies for which existing systems would require network queues implemented in the kernel space of dedicated emulation nodes.
- Our approach features configurable and modular network models. It supports complex topologies with inner routers and links, link sharing models, and overheads emulation.
- We introduce a fully decentralized monitoring algorithm for emulation of congestion, delays, and packet loss for inner nodes of the topology, without actually instantiating inner nodes nor requiring a centralized control point.

- We present support mechanisms for network emulation that enable simple selection and sharing of resources between multiple concurrent topologies and application deployments, without need for the user to directly access the physical nodes.
- We describe an implementation of our system, SplayNet, developed as an extension of the Splay [25], an open-source distributed framework that provides comprehensive facilities for the simple prototyping and deployment of networked applications and protocols.
- We evaluate our approach with several micro-benchmarks and networked applications deployed over various topologies. We compare our system to ModelNet [38] and Emulab [18, 40]. Results indicate that SplayNet achieves similar accuracy for network emulation but with lower resource requirements, and supports concurrent deployments without degradation of accuracy. Our approach scales well under heavy load and large topologies can be deployed with minimum management effort.

SplayNet is freely available as open-source software. It can be downloaded from `http://www.splay-project.org/splaynet` together with all data and source code for reproducing the experiments presented in this paper.

Outline. The paper is organized as follows. Section 2 reviews related work. Section 3 briefly introduces the open-source framework SplayNet builds upon. The design and internals of our system are described in Section 4. We present a detailed evaluation of SplayNet in Section 5 and conclude in Section 6.

2 Related Work

We classify work related to SplayNet along several perspectives, presented in Table 1. We distinguish solutions based on their operational mode (user or kernel), their need for specialized hardware or dedicated devices (switches, VLANs), and the type of orchestration for the emulation of the traffic at inner nodes/routers of the topology.[1] We also consider the support for *concurrent deployments*: multiple emulated topologies onto the same set of machines, for different users and different applications. We finally consider the ability to emulate traffic congestion along routing paths, as well as end-to-end bandwidth, delay, and packet loss. Although hardware-only emulation systems exist [21], in the remainder of this section we focus on solutions that operate partly or entirely in software. We do not consider emulators specializing in wireless networks [22, 43], nor do we focus on simulation tools [37].

ModelNet [38] uses a set of dedicated machines organized in a cluster, called *emulator nodes*. These nodes are in charge of shaping all the traffic emitted and received by the *edge nodes* supporting the application. ModelNet requires modifying the routing tables of the kernel at edge nodes to redirect all outgoing traffic toward emulator nodes. Traffic shaping rules (bandwidth and delay) are

[1] *Centralized* means that a single node is in charge of emulating the traffic for a given inner link, while different machines may be in charge of emulating different inner nodes. *Decentralized* on the other hand means that several nodes coordinate for emulating the same inner link.

Table 1. Classification of network emulation tools (B/D/P=bandwidth/delay/packet loss emulation)

Name	Mode	HW Sup.	Orchestr.	Concur. deploy.	Path congest.	Emul. B	Emul. D	Emul. P
ModelNet [38]	Kernel	×	Centralized	×	√	√	√	√
Emulab [18, 40]	Kernel	√	Centralized	×	√	√	√	√
SliceTime [39]	Kernel	√	Centralized	×	√	√	√	×
Nist NET [7]	Kernel	×	Centralized	×	×	√	√	√
ACIM [33]	Kernel	×	Centralized	×	√	√	√	√
P2PLab [28]	Kernel	×	Centralized	×	×	√	√	√
IMUNES [30]	Kernel	√	Centralized	×	×	√	√	√
Netkit [29]	Kernel	×	Centralized	×	√	√	√	√
NetEm [16]	Kernel	×	*(N/A: single link emulation only)*			×	√	√
EmuSocket [1]	User	×	*(N/A: single link emulation only)*			√	√	×
MyP2P-World [34]	User	×	Centralized	×	×	√	√	√
WiDS [26]	User	×	Centralized	×	×	×	√	√
Mininet [24]	User	×	Centralized	×	×	√	√	√
SplayNet	User	×	Decentralized	√	√	√	√	√

applied to all packets by the means of DummyNet [6] pipes set up in the kernel of emulator nodes. It is possible to deploy only one emulated topology at a time. Every topology modification requires root access to the cluster for redeploying all emulator nodes and updating the kernel routing tables at edge nodes.

Emulab [18, 40] is a shared platform that runs experiments on a dedicated emulation testbed. Although Emulab allows users to deploy several experiments under different network conditions, once a machine of the testbed is assigned to an experiment it cannot be used for any other. Emulab uses the same mechanisms as ModelNet [38] to shape traffic. To reduce the number of host machines required by each experiment, Emulab supports an *end-node-traffic-shaping* mode: the application's nodes shape the outgoing traffic themselves, relying on *tc* [16] or DummyNet [6] for, respectively, Linux- and BSD-based experiments.

Some network emulation tools are based on virtual machine deployment utilities. SliceTime [39] solves the time-drifting problem for large-scale experiments by providing a synchronization component to the deployed virtual machines. It relies on the Xen hypervisor [3]. SplayNet does not require the use of a hypervisor on the host machines, it only spawns new user-space processes to accommodate concurrent experiments.

P2PLab [28] relies on DummyNet mechanisms built in a BSD kernel. It organizes emulated networks in subnets. Each physical machine in a P2PLab cluster is responsible for a subnet and manages all the traffic within this subnet. Along the same lines, IMUNES [30] operates through a set of virtual machines interconnected via DummyNet pipes. Its originality resides in the management of the cluster hosting the virtual machines, which is driven by a peer-to-peer protocol. The protocol monitors the state of the machines and notifies the other nodes about failures and load conditions. This information is subsequently used when dispatching virtual machines. Network emulation itself operates similarly

to other DummyNet-based emulators, and it requires the physical network hosting the experiments to provide programmable VLAN support.

Mininet [24] uses lightweight virtualization mechanisms to emulate software-defined networks on a *single host*. In contrast, SplayNet and the systems presented above target deployments onto a cluster of networked machines, allowing computationally intensive tasks and greater scalability. Other low-level tools aim at shaping the traffic originated by user-space processes. Trickle [12] is a user-space bandwidth shaper for unmodified Unix applications. DelayLine [19] requires the target program to statically link against traffic-shaping libraries. The authors of [1] and [34] both propose user-space emulation tools targeting P2P protocols implemented in Java: the latter provides bytecode-level compatibility with existing applications, whereas the former offers specialized APIs. These systems only support emulation of end-to-end links characteristics and not of complete topologies, thus categorizing them as traffic shapers rather than topology emulators.

In [31], the authors propose to deploy distributed rate limiters (DRL) for general purpose cloud services. Rate limiter nodes synchronize through a lightweight UDP protocol, which shares similarities with our decentralized congestion monitoring approach (Section 4.3). DRL does not provide any support for rate-limiting multiple services concurrently running on the same nodes. SplayNet provides a per-destination dedicated token bucket, while DRL mimics the behavior of a centralized token bucket algorithm at each rate limiter node.

The support of concurrent deployments requires appropriate resource selection mechanisms. Since physical network links will be shared by multiple emulated links, the resource selection must ensure that the capacity of the physical link is sufficient for all emulated links. No emulators feature such capabilities, and most require to deploy topologies on distinct sets of nodes, thus greatly impairing scalability. The few systems that support concurrent deployments on the same nodes leave to the user the responsibility of provisioning sufficient physical capacity for emulated links.

The present work represents the first attempt to propose user-space network emulation within an integrated distributed systems evaluation framework. It provides support for concurrent deployments while offering comparable performances to single-topology and kernel-space solutions, as will be shown in Section 5.

3 Background

We implement the contributions presented in this paper as an extension to the Splay [25] open-source distributed systems evaluation framework. We chose to build upon Splay as it allows to quickly prototype, deploy, and manage distributed experiments. We present in this section some background information about Splay. We note, however, that our contributions are not specific to Splay. User-level network emulation techniques presented in this paper are applicable to other systems and platforms.

Splay's goal is to ease rapid prototyping and development of distributed protocols. It features a concise and easy-to-learn language based on Lua (`www.lua.org`). The associated libraries support the functionalities that are typically required to

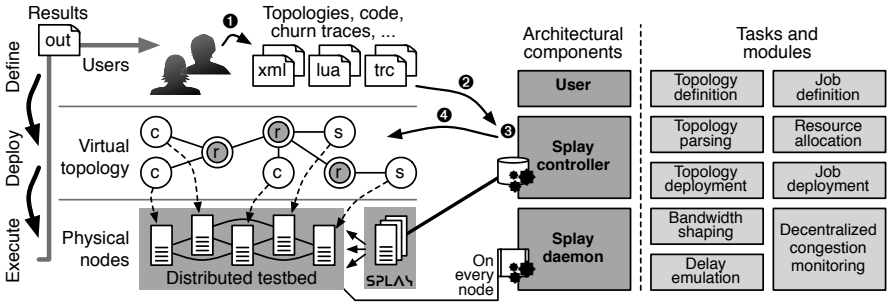


Fig. 1. The SplayNet architecture

implement distributed algorithms. The language and libraries allow implementations to be comparable in size (i.e., lines of code) to pseudo-code descriptions. This feature is on par with our objective of making network-emulated experiments and prototyping simple and fast. An example given in [25] is the Chord DHT [35]. A running implementation uses 58 lines of code, comparable in size with the pseudo-code in the original paper [35]. We use this implementation as an example application in our evaluation (Section 5). We note that the use of Lua also allows using existing code (e.g., C-based), by embedding it as a library, although we did not need to use this feature for our evaluation.

Splay also supports our objective of simplifying the usage of a testbed by providing simple multi-user resource management and deployment support. Splay runs a set of Splay *daemons* (`splayd`s) on every node of the testbed. These daemons are deployed once, by the testbed administrator. They implement sandboxing by controlling and restricting usage to resources on the nodes. This is useful in a non-dedicated environment. A single access point, the Splay *controller* (`splayctl`), orchestrates the deployment of applications. It is the sole point of access to the system for users, who do not need to have administrative access or user accounts for the machines of the testbed. The `splayctl` allows users to select nodes for deploying an application according to various criteria, and dispatches the code to the corresponding `splayd`s. The experiment is monitored and managed directly from the `splayctl`. The `splayctl` allows fine grain control of the experiments, for instance by replaying a *churn trace* that describe the dynamics of the system and is replayed by each of the `splayd`s participating to the experiment, individually for each user and for each experiment. Our approach to topology emulation is inspired by this mechanism: a topology is provided by the user along with her code and is dispatched by the `splayctl` to all selected `splayd`s part of the emulation. We describe these mechanisms and their integration in the next section.

4 The SplayNet Architecture

In this section we describe the various components necessary for supporting user-space network emulation and their integration in our SplayNet prototype. Figure 1 presents an overview of the implementation.

```
(...)
<edge src="1" dbl_kbps="4096"
      dest="4"  delayms="70"/>
<edge src="4" dbl_kbps="1024"
      dest="5"  delayms="50"/>
<edge src="5" dbl_kbps="400"
      dest="4"  delayms="50"/>
(...)
```

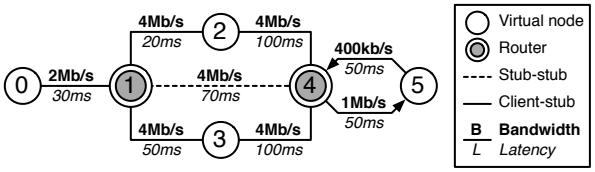


Fig. 2. Graphical representation of a topology and excerpt of its description in XML

4.1 Topology Definition and Parsing

The first step is to define a network topology to emulate. Users write an abstract description that maps vertices and edges of an undirected cyclic graph to the physical connections of a network (Figure 1-❶). Users can specify the interconnections between nodes and routers, as well as the physical properties of the links (delays, bandwidth, and packet loss rate). Application nodes can be inner nodes in the topology (and not only end-nodes), in order to support relay-based applications such as coding [15] or in-network aggregation [11]. SplayNet supports two topology description formats: the ModelNet XML-based language [42] and the Emulab TCL-based language, itself based on the one used by the NS-2 network simulator [14]. A sample topology and an excerpt of its description in XML are given by Figure 2.

The second step is the deployment (Figure 1-❷). The user submits to the Splay controller the topology description, the code to execute, and any additional files required to drive the experiments. SplayNet's topology parser extracts the graph topology. Links in the topology description are uni-directional. Non-connected topologies are rejected. The user can however request implicit link symmetry: when there is no link between two elements but a corresponding reverse link exists, an implicit link can be created, with the same characteristics as the reverse one. This operation does not modify any of the links present in the original topology, thus supporting topologies where both symmetric and asymmetric links coexist. We then use an all-pairs-shortest-path algorithm based on links delays[2] and, for every shortest path, derives the maximum available bandwidth along the path (link with the lowest bandwidth), the overall delay (sum of the delays of individual links), and the packet loss probability (product of the packet loss of individual links).

4.2 Resource Allocation and Deployment

SplayNet allocates testbed resources for executing the user code on the emulated topology (Figure 1-❸). In the context of Splay, this problem corresponds to selecting a minimal set of `splayd`s for executing the job. The allocation procedure ensures that the deployment of a topology does not impair on the accuracy of other deployed topologies, by avoiding saturating the bandwidth of

[2] Upon tie, we select a random link to balance the load but other strategies are possible, e.g., link with minimum latency or maximum bandwidth.

physical links beyond a safety margin. Finding a *minimal* set that satisfies all constraints on a shared infrastructure is a NP-hard problem [32]. Although efficient heuristics are known [10,32,41], they require knowing the start and duration of all experiments in advance, a requirement that is not met in our context. In SplayNet, we adopt a simple greedy approach to guide the selection of `splayd`s. The objective is that all links in all emulated topologies are supported by physical links with enough available capacity. We do not consider delays as a selection criterion as we assume that SplayNet will be deployed in a cluster where the latencies observed on physical links are stable and much smaller than the latency requested for the emulated paths. The `splayctl` also keeps track of the current load of the machines, as part of the regular Splay operation. The administrator provides the maximal emulated bandwidth that can be emulated on a single physical link. This value depends on the cluster hardware and network. We use a value of 100 Mb/s in our experiments, as illustrated by the concurrent deployment experiment of Section 5.3. If several `splayd`s are deployed on the same physical machine, the bandwidth available to each `splayd` is a fraction of the total available bandwidth and this value must be adjusted accordingly. For a new job, we select the least loaded nodes that satisfy the connectivity requirements, i.e., that have physical links to other nodes with sufficient available capacity taking into account the topology being deployed and those already running. If no such set of `splayd`s is found, deployment is not allowed.

We only need to map *application nodes* to `splayd`s. Routers are implicitly emulated by the communication links between the edge nodes. The advantages of this approach are twofold: first, it significantly reduces the amount of resources required to emulate large topologies; second, it frees the system from the need of powerful machines dedicated to shaping the traffic at routers. ModelNet [38] adopts a similar technique to reduce the amount of resources required for emulation in its *end-to-end* mode, but it does not emulate congestion at intermediary hops under this execution mode. We emulate traffic congestion at inner nodes with a distributed protocol and a link sharing model, described in Section 4.3.

The Splay controller finally dispatches the code to be executed to the selected `splayd`s, along with the topology information required to initialize the network emulation layer (Figure 1-❹). This information is encoded with a compact marshaller that has negligible overhead on the traffic sent to the nodes. As an example, the information necessary to emulate the topology of Figure 2 adds only 430 bytes to the data sent to each `splayd` for the job deployment.

4.3 User-Space Network Emulation

SplayNet performs link and topology emulation only in user-space, and independently for the different deployed jobs on the same `splayd`. This brings a number of benefits. First, administrators do not need to have privileged access to the machines of the testbed nor to set up any hardware network infrastructure, since the emulated network layers are initialized at the application level. Second, it overcomes a common limitation of most other state-of-the-art systems by supporting the emulation of several topologies simultaneously.

Latency and Packet Loss Emulation. Links of the topology are first characterized by latency values and packet loss rates. To account for the associated delays, the `splayd` instantiates a *countdown queue* for each outgoing link of the node of the topology being emulated. Outgoing packets traverse this queue before they reach the network. A countdown timer is initialized to the link latency value when a packet enters the queue and, upon expiration, the packet is sent over the wire. Note that the actual latency of the physical topology is assumed to be orders of magnitude smaller than the emulated one, as all `splayd`s are typically executed on a cluster. Otherwise, the value of the countdown timer should be adjusted to take into account delays observed at the physical level.

The reactivity to the timer expiration is crucial for accurate emulation, especially when emulating low-latency links, thus the choice of the underlying operating system plays an important role for achieving good performance in link delay emulation. We evaluated the scheduling accuracy on various operating systems, and reproduced results on par with those presented in [13]. Scheduling accuracy is around 0.1 ms for Linux 2.6, and in the order of a few milliseconds for Linux 2.4 and FreeBSD 7.3. This indicates that accurate latency emulation is achievable, with measurable errors in the order of milliseconds.

Packet loss is enforced by simply dropping random packets at the source according to the calculated loss rate on the path to their destination. Here again, we assume that the underlying physical network has a negligible packet loss rate that we do not need to compensate.

Bandwidth Shaping. In addition to latency, a topology specifies the maximal bandwidth for each of its links. The actual bandwidth available to the application will be smaller, and depends on the size of the messages sent through the socket. Our model takes into account emulation of overhead as follows.

For TPC/IP and UDP/IP, we use the default Ethernet MTU size of 1500 B (bytes). Ethernet overheads consist of 38 B for each message: 12 B of source and destination addresses, 8 B of preamble, 14 B of header, and 4 B of trailer. We then add the overhead of IPv4 (20 B), and TCP or UDP headers (20 and 28 B, respectively). The overhead factors in the number of packets for a given application-level message, and determines the bandwidth that is actually used on the emulated link. This overhead model, which can be easily modified to account for different network settings, allows us to precisely emulate the actual bandwidth available to an application sending messages of various sizes. It is also independent from the configuration of the supporting physical network (e.g., the use of jumbo frames).

We use a token bucket algorithm [36] to cap the throughput of outgoing traffic to the value specified in the emulated topology.[3] The algorithm operates by inserting a number of tokens at a fixed rate (determined according to the available bandwidth) into a virtual bucket. Each token represents a fixed amount of bytes that can be sent. Application-level packets are delivered over the wire

[3] The *tc* [16] tool integrated in the Linux kernel uses a similar approach to bandwidth shaping. However, SplayNet is cross-platform and does not rely on any kernel support as it integrates its own shaping mechanism.

only if the corresponding amount of tokens is available in the bucket. Otherwise, they are re-queued in the bucket. This simple strategy guarantees a consistent average throughput during emulation.

The bucket fill rates are initially configured to the minimum available bandwidth across all hops on the shortest path between the source and destination nodes. Afterwards, fill rates are dynamically adjusted by the decentralized congestion monitoring protocol based on the actual available bandwidth on the path, dynamically considering other flows taking place in the topology.

Decentralized Congestion Emulation. The delay emulation and bandwidth shaping mechanisms are the foundations of a decentralized network emulation platform, and are the first components of the emulated network model. They are, however, not sufficient for accurately emulating network congestion across multi-hop routing paths. This task is the responsibility of a decentralized congestion monitoring protocol, which constitutes the second part of our model. Note that the network model is modular: both parts can be modified independently of the emulation framework, and new models can be integrated, with different overheads, link sharing, or QoS policies.

In a centralized solution such as ModelNet [38], one or a small set of dedicated hosts are continuously keeping track of the network traffic on all possible paths of the topology, since all packets are routed through these hosts. This global view of the network allows throttling the data rates according to the limits imposed by the topology.

We advocate the use of a decentralized architecture that does not require specific nodes to handle all traffic passing across the topology. Instead, we rely on a distributed protocol to promptly distribute notifications about the start and end of data streams. These notifications are disseminated to all the nodes involved in the emulation of a given topology through fast and reliable UDP multicast channels (PGM).

View update. Whenever a node starts or stops sending data using TCP, it first updates its local view of ongoing network flows by incrementing the number of competing flows on every hop from itself to the destination node. Then, it disseminates this information to the other nodes by specifying the source, the destination, and the virtual routing hops involved in the stream. In the context of a large-scale topology deployment (Section 5.4) with 150 nodes, we observe average dissemination delays of 7.36 ms. Upon receiving this information, the other nodes adjust their local view accordingly by updating the number of competing streams on affected links and, if necessary, the token bucket's fill rates. In the case of UDP streams, it is not possible to determine the end of a communication as with TCP. Hence, we adopt a periodic report strategy: every 50 ms, the amount of data sent through the socket is propagated to other nodes, which update their state based on information from the previous period.

Each node needs to maintain an up-to-date view of ongoing data flows on the emulated network, whether originated by itself or by other nodes, and determine how internal links bandwidth is shared between competing flows. This view is efficiently modeled as a n-ary tree rooted at the local node.

The leaves of the tree represent the other virtual end-nodes, while inner nodes correspond to the routers. Edges of the tree are labeled with their maximum bandwidth capacity and latency, and they embed a counter that keeps track of the number of active data streams on the associated links. Each leaf is augmented with a token bucket that specifies the maximum data rate allowed to reach the corresponding node. The initial fill rate for each token bucket corresponds to the bandwidth allowed by the path from the local node to the leaf.

Link sharing model. Whenever multiple streams share a segment of the routing path, the token bucket fill rates are adjusted to split the bandwidth between the competing streams, for each of the internal links of the topology supporting multiple streams. The split depends on a bandwidth sharing models. The basic *Max-Min* sharing model introduced in [4] does not correctly reflect actual sharing behaviors [9]. Therefore, we use the *RTT-aware Max-Min* sharing model [20,27], which is widely considered as accurate.

First, the allocation of bandwidth ρ_i for each flow f_i on a link is capped by the limitation of its bandwidth-delay product: the flow is capped by the ratio of the sending window size W_i and roundtrip RTT_i: $\rho_i \leq W_i/RTT_i$.[4] Second, the sum of ρ_i for all flows on the link must not exceed the capacity of the link F. The share ρ_i of the available bandwidth for each flow is then inversely proportional to the flow RTT_i, i.e., $\rho_i = F \times ((RTT_i)^{-1} \sum_{j=1..n} (RTT_j)^{-1})$ when the first capacity constraint does not apply to any flow. The allocation takes into account the fact that some hops in end-to-end paths are not able to use their full share of a given emulated link. In this case, the remaining bandwidth is redistributed to other existing communication flows under the model constraints until no further refinement is possible.[5]

Example. Figure 3 illustrates the tree maintained by node 0 in the topology of Figure 2 as communication flows are established between nodes. Initially, no communication takes place and the buckets are idle (first tree in the figure). Then, node 0 starts communicating with node 2. To that end, it sends to other nodes information about the path that has been estab-

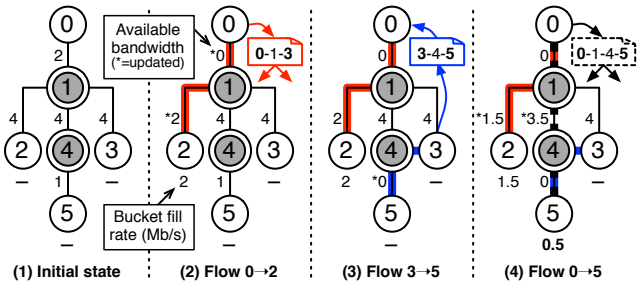


Fig. 3. Evolution of the tree maintained by node 0 for the topology of Figure 1 with the establishment of 3 communication flows

[4] We use a default value of 64 KB for the sending window size W_i, as found on most wired networked system.

[5] Note that the current model considers that the reverse-path bandwidth is sufficient to accommodate the traffic of ACKs. Refinement of the model may include these aspects, e.g., based on [17].

lished, and it updates its local view by adjusting the available bandwidth on links and the fill rate of the bucket at leaf 2 (second tree in the figure). After receiving a message from node 3 that starts sending data to node 5, node 1 simply updates the available bandwidth on the links but does not need to change the bucket fill rates as there is no competition with one of its communication flows (third tree in the figure). Finally, node 1 communicates with node 5. The new flow competes with the previous two as it shares a link with each of them: link 0→1 for the first flow, and link 4→5 for the second. The sharing of these links is determined according to the RTT-aware Min-Max sharing model and the bucket fill rate of leaves 2 and 5 are adjusted accordingly (fourth tree in the figure). From the topology description in Figure 2, we obtain the following RTTs: 0→2 is 100 ms, 3→5 is 300 ms and 0→5 is 300 ms. As a result, the 2 Mb/s of the link 0→1 are shared as 75% of 2 Mb/s = 1.5 Mb/s for 0→2, and 25% of 2 Mb/s = 0.5 Mb/s for 0→5. Note that the bandwidth allocated to flow 0→5 is the maximal allocatable, as link 4→5 is shared with flow 3→5 with the same RTT for both flows.

5 Evaluation

In this section we present an extensive evaluation of our contributions. We compare SplayNet with the *de facto* reference network emulators ModelNet [38] and Emulab [18,40]. Similarly to SplayNet, both systems provide complete emulation toolsets, from a topology description language to topology deployment facilities. We use the same application code over the three emulation systems, by using Splay and Lua stand-alone libraries on ModelNet and Emulab.

In Section 5.1 we first present a set of micro-benchmarks that measure the accuracy of the delay and bandwidth emulation on simple yet representative topologies. Our study then proceeds with a set of macro-benchmarks based on real-world applications (Section 5.2). We use the Chord DHT [35] as an example of delay-sensitive application and collaborative application-level multicast using parallel n-ary trees [5] as an example of a bandwidth-sensitive application.

One of the distinctive features of SplayNet is the support for concurrent deployments of multiple topologies on the same testbed. In Section 5.3, we investigate the scalability and accuracy of SplayNet when concurrently deploying several topologies. Finally, Section 5.4 concludes this evaluation by presenting the behavior of SplayNet when emulating large and complex topologies.

We set up a SplayNet cluster on top of a 1 Gb/s switched network with 60 machines, each with 8-Core Xeon CPUs and 8 GB of RAM. The ModelNet cluster is deployed on the same machines. We used the similarly powerful `pc3000`[6] machines for Emulab experiments.

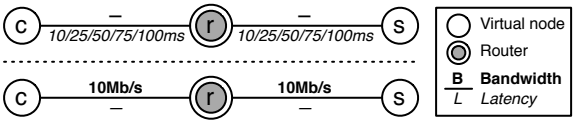
The SplayNet modules executed by the `splayd`s for network shaping are implemented in pure Lua. We use version 5.1.4 of the Lua virtual machine for all the experiments. The `splayctl` extensions are implemented in Ruby. Due to the small number of machines typically available on Emulab, we had to restrict our evaluations on this platform to a maximum of 20 nodes per experiment.

[6] `emulab.net/shownodetype.php3?node_type=pc3000`

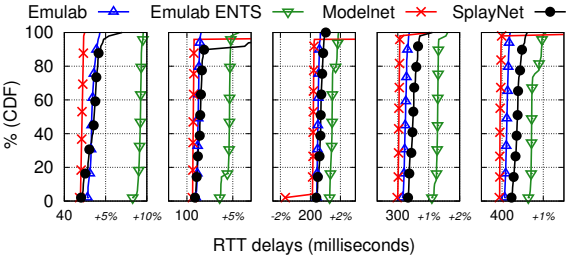
5.1 Micro-Benchmarks

Latency. To evaluate the accuracy of link latency emulation, we deploy a simple client-server application using remote procedure calls (RPCs) at the edges of the topology, as shown in Figure 4.a (top). We measure the accuracy of the RPC's round-trip-time (RTT) for increasing emulated latencies. This experiment also includes results for Emulab configured in end-node-traffic-shaping (ENTS) mode to remove any latency overhead toward a third-party shaping node.

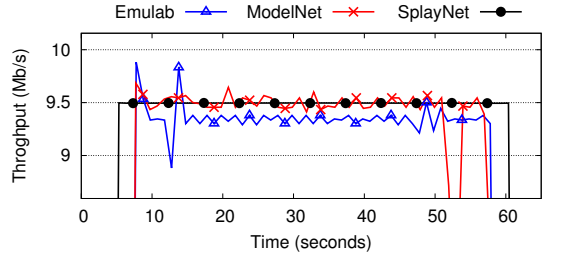
Figure 4.b presents the cumulative distribution function (CDF) of observed delays. The expected RTT is shown on the x-axis for each of the link latency values, with variations expressed as percentages. Performance over the 3 testbeds is very similar: emulated latencies never deviate more than 10% from the expected values, and never more than 5 milliseconds in absolute terms.



(a) Topologies: latency (top) and bandwidth (bottom).



(b) Link latency emulation.

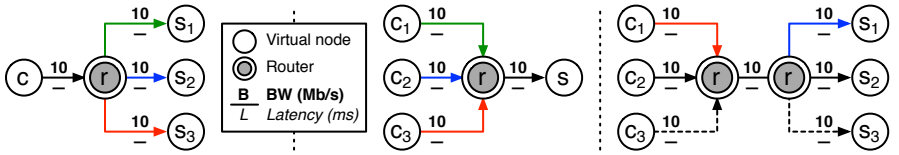


(c) Link bandwidth emulation.

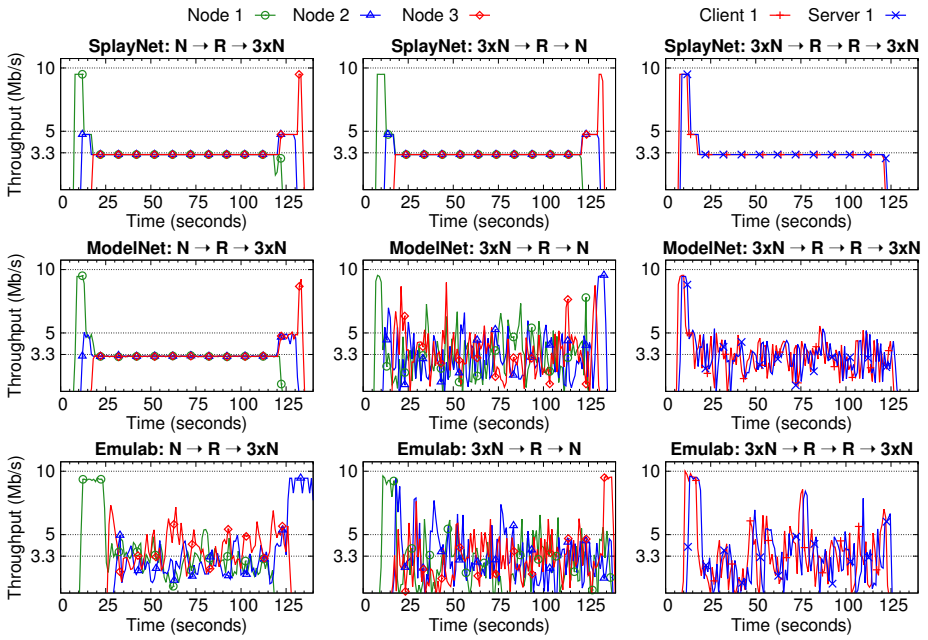
Fig. 4. Link latency and bandwidth emulation for a client-server RPC benchmark

Bandwidth. Our second micro-benchmark evaluates the accuracy of the bandwidth emulation. We deploy the point-to-point topology of Figure 4.a (bottom) with two nodes connected by a single router. Link latencies are close to zero (bare latencies of the support cluster) to mitigate any bandwidth-delay-product effect [20,27] and to allow the maximum theoretical throughput. Emulab and ModelNet's link queue sizes are configured to the default size of 100 slots.

The client node continuously streams data to a server over a 10 Mb/s link via a pre-established TCP connection. Figure 4.c shows how the three systems let the application-level data stream, and emulated overhead, saturate the available link bandwidth up to the theoretical limits. ModelNet and Emulab present oscillations in the observed instantaneous throughput, while SplayNet provides a more steady download rate. This is a result of our choice of a decentralized, model-based network emulation that does not use kernel-level buffers at dedicated nodes. Oscillations are observed in real networks but to a much smaller



(a) Topologies for bandwidth emulation micro-benchmarks (N→R→3N, 3N→R→N, 3N→R→R→3N).



(b) Observed throughput at client nodes.

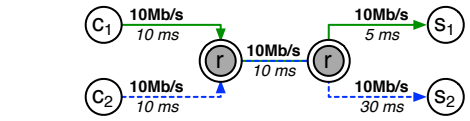
Fig. 5. Bandwidth shaping accuracy

extent than with ModelNet and Emulab. For the range of application of SplayNet (evaluation of networked protocols), the current model allows reproducibility between runs and between applications. We emphasize that oscillatory bandwidth allocation or reverse ACK traffic [17] can be integrated in the model without re-engineering the other elements of SplayNet.

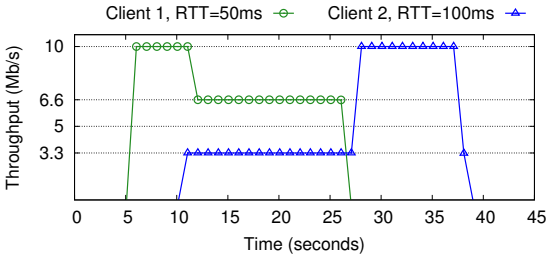
We deploy more complex scenarios in order to evaluate the accuracy of SplayNet's bandwidth emulation when multiple clients concurrently stream data through common intermediate nodes. We use three topologies shown in Figure 5.a. Nodes are linked via 10 Mb/s links. Client nodes stream 50 MB of data to server nodes, competing for the bandwidth on the link that connects the client to the router (topology on the left, labeled N→R→3N), the link that connects the router to the server (topology on the center, labeled 3N→R→N), or the link between the two router nodes (topology on the right, labeled 3N→R→R→3N). Streams are started at intervals of 5 seconds. For the sake of clarity, in the case of 3N→R→R→3N, we only present the observed throughput at one client and

one server. In order to isolate bandwidth emulation evaluation from the sharing model, we consider equal delays on all links (bare delay from the underlying network). The observed throughput in Figure 5.b indicates that SplayNet provides each stream with a fair amount of bandwidth even when competing with other streams, without dedicated machines to emulate routers and with no centralized traffic shaping orchestration. The results obtained with ModelNet and Emulab provide the applications an average throughput that is reasonably close to the expected value, but they are hardly reproducible from one run to another or over the duration of an experiment.

Link Sharing. We now evaluate the effectiveness of the RTT-aware Max-Min link sharing model introduced in Section 4.3. We use the topology described by Figure 6.a and set up two flows from c_1 to s_1 and from c_2 to s_2. The r→r link is shared by the two flows and the maximal bandwidth achievable by both due to their bandwidth-delay product is greater than the link's capacity of 10 Mb/s. The first flow starts at second 5 while the second starts at second 10. As expected, when the intermediate link is traversed by both flows, its capacity is split according to the inverse of each flow's RTT: $\frac{50}{150} = \frac{2}{3}$ of 10 Mb/s for client 1 (∼6.66 Mb/s), and the remaining $\frac{1}{3}$ of 10 Mb/s for client 2 (∼3.33 Mb/s).



(a) Topology.



(b) Observed bandwidth at servers s_1 and s_2.

Fig. 6. RTT-aware Max-Min sharing of 10 Mb/s bottleneck link

5.2 Macro-Benchmarks

For our set of macro-benchmarks, we deploy complete implementations of two representative distributed protocols, for which network emulation can be instrumental to evaluate the performance and behavior. For both experiments, using the Chord DHT [35] and a collaborative multicast application [5], nodes are deployed on an emulated star topology where all end-nodes are connected through a single central inner router. All links from the end-nodes to the router are emulated at 10 Mb/s (symmetric) with 30 ms latency.

Delay-sensitive: Chord DHT. Our first representative protocol is the Chord DHT [35]. After 20 nodes form a stabilized Chord ring, each node submits 50 queries for random keys. Note that the constructed rings do not perfectly overlap due to the nature of Chord node identifiers. In particular, node identifiers are initialized by hashing their IP and port, and Emulab does not allow choosing the network mask of the assigned machines.

Figure 7 presents the CDF of the delays for all queries (left) and the CDF of the number of *hops* required by the queries to reach the node in charge of the key (right). The results demonstrate similar behavior across all the testbeds in terms of latency emulation.

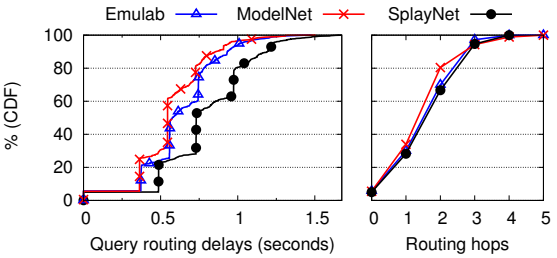


Fig. 7. Routing in a 20 nodes Chord ring

Bandwidth-sensitive: multicast. We now evaluate how SplayNet performs compared to ModelNet and Emulab for bandwidth-intensive protocols. We use a multicast protocol based on parallel n-ary trees [5]. We create n=4 distinct trees as done in SplitStream [8]. Each of the 20 nodes is an inner member in one tree and a leaf in the others. The data to transmit is split into 16 blocks of 2.5 MB each. Blocks are propagated in parallel along the 4 trees using a round-robin policy for tree selection.

Figure 8 presents the CDF of the download completion time for all 4 trees at all nodes. The results indicate that the three platforms offer comparable performance in terms of bandwidth emulation.

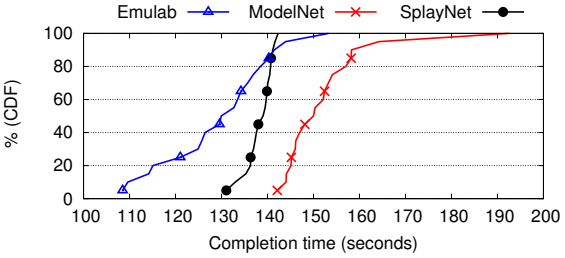


Fig. 8. Multicast diffusion on n-ary trees

5.3 Concurrent Deployments

We now evaluate the impact of concurrent deployments in the same testbed on the emulation accuracy for both delay- and bandwidth-sensitive protocols. In these experiments, we use only 10 physical nodes of our cluster to enforce a high level of concurrency. Each individual deployment consists of 20 nodes in a star-like topology with 30 ms latency and 10 Mb/s bandwidth links. In the most extreme case of 50 concurrent jobs, up to 1,000 application nodes run simultaneously on the testbed. We start with a delay-sensitive application.

Figure 9 presents the results of query routing delays when deploying up to 50 concurrent jobs, each running one instance of the Chord DHT. Each bar in a group of four presents a representative percentile (the first quartile, the median, the 90th and 99th percentile) of the routing delays for 50

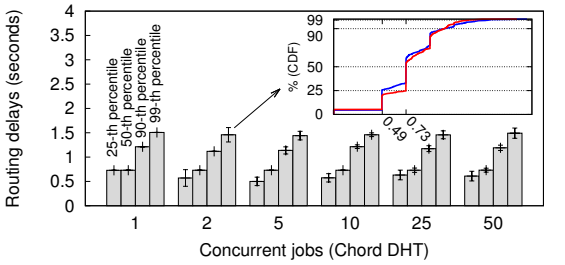


Fig. 9. Impact of concurrent deployments on delay-sensitive protocol Chord

random queries issued by the nodes. The inner graph shows the CDF of the routing delays for the queries issued by the nodes in the case of two concurrently deployed jobs, for which the percentiles give a compact representation. The standard deviation for each quantile is indicated on each bar.

For instance, the median routing delay for two concurrent experiments are 0.49 s and 0.73 s, yielding an average median of 0.61 s and a standard deviation of 0.17 s. The small standard deviations and consistent quantiles confirm the lack of variation between the observed performances of concurrently deployed jobs.

We continue by performing multiple concurrent deployments of a bandwidth-sensitive protocol, the parallel n-ary tree protocol previously described. Our objective is that concurrent experiments have little to no impact on one another, and in particular on the behavior of the protocol under test. The behavior of a set of protocols is represented by the CDF of the completion time for retrieving a file from the parallel trees. We use a star topology with low and high bandwidth requirements.

In low bandwidth settings, each link in the topology supports a bandwidth of 128 Kb/s and the transmitted file size is 2 MB. We observe in Figure 10 that the deployment of 1 to 50 concurrent instances of the protocol have no impact on their performance, allowing to safely rely on a shared emulation testbed. The emulated traffic passing through each physical link of the cluster is below the threshold of 100 Mb/s we use in our experiments.

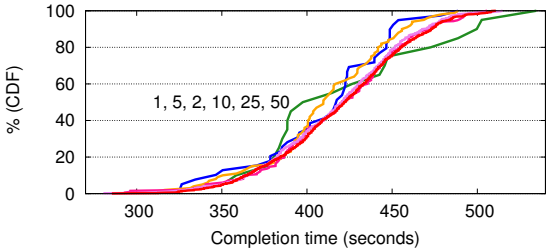


Fig. 10. Concurrent n-ary tree deployments: 2 MB of data with 128 Kb/s links (the number of concurrent deployments is indicated next to the respective lines)

We also experimented in high bandwidth settings, with 10 Mb/s links in the emulated topology and a file size of 40 MB, and observed consistent behavior of the protocols and topologies from 1 to 5 concurrently deployed topologies. With more, as expected, concurrent deployments adversely impact one another due to the maximal emulated traffic of 100 Mb/s per physical link.

5.4 Scalability

In this last experiment, we evaluate the accuracy and scalability of SplayNet when emulating large and complex topologies. We compare the accuracy of the emulation against "ideal" results obtained using a centralized and omniscient simulation. Based on the full list of exchanges, we determine the exact congestion on inner links and decide on appropriate bandwidth allocation with no synchronization delay. The simulation uses the same mechanisms for deciding on bandwidth allocation (Section 4.3) but applies them to the full topology graph. We use a set of three topologies of size 50, 100, and 150 nodes, constructed using

the *preferential attachment* method [2]. We start with a single node and add new nodes one by one, each with one outgoing link. We pick the destination of that link such that the selection probability is proportional to each node's actual in-degree. This method yields *scale-free* networks, representative of the characteristics of Internet topologies, with distribution of the degrees following a power-law. Nodes with no incoming link act as application nodes while other nodes are routers. Due to the scale-free nature of the graph, a large majority of paths between end-nodes share common inner links in the topology. This is a challenge for the distributed congestion evaluation mechanism. Each link has a random delay in the [10:30] ms range. Bandwidth between routers is 1 Mb/s, and 10 Mb/s from end-nodes to their respective routers, to prevent the last link be a bottleneck and to emphasize the effect of congestion on inner links.

We mimic a randomized bandwidth-sensitive communication workload. Some application nodes initiate a single communication of 10 MB of data over TCP to a randomly selected other node. This is similar to what would happen for instance in a BitTorrent dissemination. For each topology, we use two workloads: a light and a heavy one (first and second line of Table 2, respectively), which differ in particular in the number of (concurrent) exchanges. The last four columns present the statistics for the accuracy, that is, the variation over the ideal simulation for the same exchanges. The average accuracy ranges from ±1.02% to ±11.94%, with only small variations across all flows and in all cases, i.e., a low standard deviation. Minimal and maximal inaccuracy is particularly low for the smallest graph and remains reasonable for the two others, well in the usability range for large-scale network emulation. We were not able to deploy the same experiment on Emulab due to the low number of available nodes on this platform.

Table 2. Accuracy versus centralized simulation, on large scale-free topologies, of a randomized high-bandwidth communication workload

		flows/s		accur. error (±%)			
nodes	**routers**	**avg.**	**time**	**avg.**	**stdev.**	**min.**	**max.**
30	20	4.54	398.92 s	1.02	0.92	0.04	2.61
		9.78	719.11 s	3.89	2.14	0.08	8.15
62	38	7.49	400.85 s	3.45	2.12	0.65	8.52
		15.34	959.56 s	5.63	3.38	1.46	17.12
98	52	9.79	566.56 s	4.00	1.80	1.83	7.68
		19.09	1201.38 s	11.94	4.75	0.23	24.48

6 Conclusion

Network emulation allows researchers to evaluate distributed applications by deploying them in a variety of network conditions. Previous solutions often relied on dedicated machines to shape the network traffic across the nodes involved in an experiment, and did not allow the concurrent deployment of different network topologies on the same nodes of a testbed.

This paper introduced SplayNet, an integrated user-space network emulation framework. SplayNet uses a distributed orchestration protocol to emulate congestion at inner nodes in a decentralized manner and without instantiating

these inner nodes on physical machines. It allows the deployment of multiple experiments, each under different network emulation conditions, and running concurrently on the same set of machines. SplayNet offers equivalent performance to state-of-the-art systems, both in terms of latency emulation and bandwidth shaping accuracy. It has shown to scale well for concurrent deployments of real-world distributed protocols and large topologies. This work was partly supported by the Swiss National Foundation under agreement number 200021-127271/1.

References

1. Avvenuti, M., Vecchio, A.: Application-level network emulation: the EmuSocket toolkit. Journal of Network and Computer Applications 29(4) (2006)
2. Barabasi, A.-L., Albert, R.: Emergence of scaling in random networks. Science 286 (1999)
3. Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I., Warfield, A.: Xen and the art of virtualization. In: SOSP (2003)
4. Bertsekas, D., Gallager, R.: Data Networks. Prentice-Hall (1992)
5. Biersack, E.W., Rodriguez, P., Felber, P.: Performance analysis of peer-to-peer networks for file distribution. In: Solé-Pareta, J., Smirnov, M., Van Mieghem, P., Domingo-Pascual, J., Monteiro, E., Reichl, P., Stiller, B., Gibbens, R.J. (eds.) QofIS 2004. LNCS, vol. 3266, pp. 1–10. Springer, Heidelberg (2004)
6. Carbone, M., Rizzo, L.: Dummynet revisited. SIGCOMM Comput. Commun. Rev. 40(2), 12–20 (2010)
7. Carson, M., Santay, D.: NIST Net-a linux-based network emulation tool. SIGCOMM Comput. Commun. Rev. 33(3), 111–126 (2003)
8. Castro, M., Druschel, P., Kermarrec, A.-M., Nandi, A., Rowstron, A., Singh, A.: Splitstream: high-bandwidth multicast in cooperative environments. In: SOSP (2003)
9. Chiu, D.M.: Some observations on fairness of bandwidth sharing. In: ISCC (2000)
10. Coffman Jr, E., Garey, M., Johnson, D.: Approximation algorithms for bin packing: A survey. In: Approximation algorithms for NP-hard problems, pp. 46–93. PWS Publishing Co. (1996)
11. Costa, P., Donnelly, A., Rowstron, A., O'Shea, G.: Camdoop: exploiting in-network aggregation for big data applications. In: NSDI (2012)
12. Eriksen, M. Trickle: A userland bandwidth shaper for unix-like systems. USENIX ATC (2005)
13. Etsion, Y., Tsafrir, D., Feitelson, D.: Effects of clock resolution on the scheduling of interactive and soft real-time processes. In: SIGMETRICS (2003)
14. Fall, K.: Network emulation in the Vint/NS simulator. In: ISCC (1999)
15. Gkantsidis, C., Rodriguez, P.: Network coding for large scale content distribution. In: INFOCOM (2005)
16. Hemminger, S.: Network emulation with NetEm. In: Linux Conference (2005)
17. Heusse, M., Merritt, S.A., Brown, T.X., Duda, A.: Two-way tcp connections: old problem, new insight. SIGCOMM Comput. Commun. Rev. 41(2), 5–15 (2011)
18. Hibler, M., Ricci, R., Stoller, L., Duerig, J., Guruprasad, S., Stack, T., Webb, K., Lepreau, J.: Large-scale virtualization in the emulab network testbed. USENIX ATC (2008)
19. Ingham, D.B., Parrington, G.D.: Delayline: a wide-area network emulation tool. Comput. Syst. 7(3), 313–332 (1994)
20. Kelly, F.P.: Charging and rate control for elastic traffic. European Trans. on Telecommunications 8, 33–37 (1997)

21. Kodama, Y., Kudoh, T., Takano, R., Sato, H., Tatebe, O., Sekiguchi, S.: GNET-1: Gigabit ethernet network testbed. In: CLUSTER (2004)
22. Kojo, M., Gurtov, A., Manner, J., Sarolahti, P., Alanko, T., Raatikainen, K.: Seawind: a wireless network emulator. In: MMB (2001)
23. Kristiansen, S., Plagemann, T., Goebel, V.: Towards scalable and realistic node models for network simulators. In: SIGCOMM (2011)
24. Lantz, B., Heller, B., McKeown, N.: A network in a laptop: rapid prototyping for software-defined networks. In: HotNets (2010)
25. Leonini, L., Rivière, E., Felber, P.: SPLAY: Distributed systems evaluation made simple. In: NSDI (2009)
26. Lin, S., Pan, A., Zhang, Z., Guo, R., Guo, Z.: WiDS: an integrated toolkit for distributed system development. In: HotOS (2005)
27. Massoulié, L., Roberts, J.: Bandwidth sharing: objectives and algorithms. IEEE/ACM Trans. Netw. 10(3), 320–328 (2002)
28. Nussbaum, L., Richard, O.: Lightweight emulation to study peer-to-peer systems. Concur. and Comput.: Practice and Experience 20(6), 735–749 (2008)
29. Pizzonia, M., Rimondini, M.: Netkit: easy emulation of complex networks on inexpensive hardware. In: TridentCom (2008)
30. Puljiz, Z., Penco, R., Mikuc, M.: Performance analysis of a decentralized network simulator based on IMUNES. In: SPECTS (2008)
31. Raghavan, B., Vishwanath, K., Ramabhadran, S., Yocum, K., Snoeren, A.: Cloud control with distributed rate limiting. SIGCOMM Comput. Commun. Rev. 37, 337–348 (2007)
32. Ricci, R., Alfeld, C., Lepreau, J.: A solver for the network testbed mapping problem. SIGCOMM Comput. Commun. Rev. 33(2), 65–81 (2003)
33. Ricci, R., Duerig, J., Sanaga, P., Gebhardt, D., Hibler, M., Atkinson, K., Zhang, J., Kasera, S., Lepreau, J.: The Flexlab approach to realistic evaluation of networked systems. In: NSDI (2007)
34. Roverso, R., Al-Aggan, M., Naiem, A., Dahlstrom, A., El-Ansary, S., El-Beltagy, M., Haridi, S.: MyP2PWorld: Highly reproducible application-level emulation of P2P systems. In: SASOW (2008)
35. Stoica, I., Morris, R., Liben-Nowell, D., Karger, D.R., Kaashoek, M.F., Dabek, F., Balakrishnan, H.: A scalable peer-to-peer lookup protocol for internet applications. IEEE/ACM Trans. Netw. 11(1), 17–32 (2003)
36. Tang, P., Tai, T.: Network traffic characterization using token bucket model. In: INFOCOM (2009)
37. Tazaki, H., Asaeda, H.: DNEmu: Design and implementation of distributed network emulation for smooth experimentation control. In: Korakis, T., Zink, M., Ott, M. (eds.) TridentCom 2012. LNICST, vol. 44, pp. 162–177. Springer, Heidelberg (2012)
38. Vahdat, A., Yocum, K., Walsh, K., Mahadevan, P., Kostic, D., Chase, J., Becker, D.: Scalability and accuracy in a large-scale network emulator. In: OSDI (2002)
39. Weingärtner, E., Schmidt, F., Lehn, H., Heer, T., Wehrle, K.: SliceTime: a platform for scalable and accurate network emulation. In: NSDI (2011)
40. White, B., Lepreau, J., Stoller, L., Ricci, R., Guruprasad, S., Newbold, M., Hibler, M., Barb, C., Joglekar, A.: An integrated experimental environment for distributed systems and networks. In: OSDI (2002)
41. Yin, Q., Roscoe, T.: VF2x: Fast, efficient virtual network mapping for real testbed workloads. In: Korakis, T., Zink, M., Ott, M. (eds.) TridentCom 2012. LNICST, vol. 44, pp. 271–286. Springer, Heidelberg (2012)
42. Zegura, E., Calvert, K., Bhattacharjee, S.: How to model an internetwork. In: INFOCOM (1996)
43. Zheng, P., Ni, L.: Empower: A network emulator for wireline and wireless networks. In: INFOCOM (2003)